임베디드 소프트웨어 개발의 12가지 기본 사항

첫 단계부터 마스터 클래스로 가기 위한
당신에게 필요한 사례 연구

차례

우리는 임베디드 시스템 개발을 위한 소프트웨어 및 서비스를 제공하는 세계 선두기업으로, 고객의 현재 제품과 미래의 혁신을 만들고 보호할 수 있도록 지원합니다.

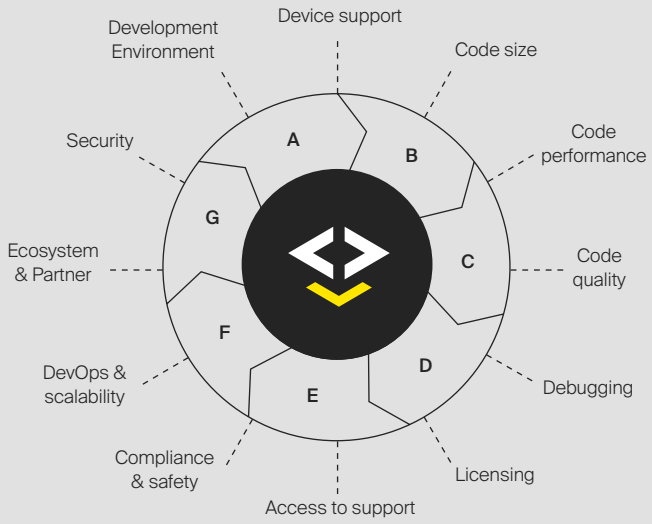
IAR Systems와 서비스, 제품에 대한 자세한 내용은 iar.com 에서 확인 가능합니다.

자신 있는 품질로 출시 시간(Time To Market) 단축

출시 시간을 단축하고 품질을 확보하는 동시에 예산 범위를 충족하는 것이 가능할까요? 특히 제품 차별화를 통해 시장에서 제품 성공을 보장하는 임베디드 소프트웨어 개발에 있어서 기업은 어떤 투자가 명확한 ROI(Return on Investment, 투자 수익)와 합리적인 TCO(Total Cost of Ownership, 총 소유 비용)로 귀결되는지 따져봐야 합니다.

그림 1. IAR Systems의 시각에서 본 임베디드 소프트웨어 개발의 기본 사항

(출처: IAR Systems)



A Global technical support
B Debugging & trace probes
C Static code analysis
D Runtime analaysis
E Safety Certification
F IP protection
G Production control

소비재 및 통신 제품에서 자동차 애플리케이션과 산업용 장비에 이르기까지 고객은 점점 더 짧은 주기로 제품에 대해 더 많은 새로운 기능을 요구합니다. 시장의 이러한 요구는 제품의 차별화와 개발에 중요한 임베디드 소프트웨어에 직접적인 영향을 미칩니다. 임베디드 애플리케이션은 그 어느 때보다 복잡해 졌으며 다양한 기술을 가진 대규모의 분산된 팀에서 구축됩니다. 임베디드 애플리케이션에 대한 요구 사항을 충족하기 위해 해결해야 할 많은 과제와 우려가 있지만 시장에서 차별화를 위해 개발자는 여전히 혁신에 집중하고 제품을 최대한 활용할 수 있어야 합니다.

지난 40년 동안 IAR Systems는 임베디드 소프트웨어 개발자의 일상적인 작업 과정의 일부였으며 그들의 프로세스를 깊이 이해하고 있습니다. IAR Systems의 시각에서 보았을 때 임베디드 소프트웨어 개발의 기본사항(그림 1)은 개발할 제품의 생산성, 효율성 및 품질 뿐만 아니라 비용과 출시 시간에도 영향을 미칩니다.

임베디드 소프트웨어 개발의 총 12가지 기본 사항은 다음과 같이 확장될 수 있습니다.

01 **개발 환경**: 프로젝트 관리 툴과 편집기가 하나로 통합된IDE가 바람직합니다.

02 **디바이스 지원**: 여러 공급업체의 8, 16, 32 및 64비트 코어의 다양한 디바이스를 지원하여 요구 사항이 다른 다양한 프로젝트를 동시에 사용이 가능합니다.

03 **코드 크기**: 애플리케이션 최적화를 통해 더 작은 디바이스를 사용할 수 있어 비용을 절감할 수 있습니다.

04 **코드 성능**: 더 빠른 코드와 더 나은 사용자 경험이 가능합니다.

05 **코드 품질**: 최고의 프로그래밍 방식을 사용하여 더 나은 제품을 구현합니다.

06 **디버깅**: 버그를 제거하고 품질을 향상시키기 위해 애플리케이션을 실시간으로 완전히 제어할 수 있는 것이 핵심입니다.

07 **라이선싱**: 손쉬운 라이선스 관리를 통해 고객이 단일 사용자에서 라이선스 풀에 이르기까지 필요에 따라 정확한 관리와 합당한 비용을 지불할 수 있습니다.

08 **지원에 대한 액세스**: 프로그래머가 코드에 집중하고 필요할 때 지원과 교육을 받을 수 있도록 하는 것이 필수입니다.

09 **DevOps 및 확장성**: 증가하는 수요를 해결하고 조직 인프라를 자동화된 CI/CD 워크플로우로 현대화합니다.

10 **규정 준수 및 안전**: 회사가 해당 부문의 특정 요구 사항을 준수하고 있음을 증명해야 합니다.

11 **생태계 및 파트너**: 고객에게 혜택을 주고 새로운 디바이스, 미들웨어 및 통합이 향후 지원될 것이라는 확신을 제공합니다.

12 **보안**: 필수적인 것으로 기업은 개발 초기 및 후반 단계에서 보안을 구현하는 방법을 모색하고 있습니다.

IAR Systems의 임베디드 개발 솔루션은 임베디드 소프트웨어 개발의 모든 기본 사항을 포함하고 있습니다. 이는 생산성과 효율성을 향상시켜 제품 품질을 확보하며 출시 시간을 단축하는 가치를 더합니다.

이 모든 것에서는 비용이 수반되며 ROI 및 TCO 사용 사례에서 확인할 수 있습니다. 다음은 기본 사항에 대한 고려를 통해 ROI 및 TCO에 긍정적인 영향을 미치는 구체적인 사례입니다. 보안은 완전히 별도로 분석해야 하므로 사례 연구에서 다루지 않습니다. “생태계 및 파트너”와 “디바이스 지원” 주제는 “개발 환경” 사용 사례에서 다룹니다.

1. 코드 크기

코드 크기와 벤치마크에 유의해야 하는 이유는 무엇일까요?
코드 크기를 작게 유지함으로써 주어진 디바이스에 더 많은 기능을 구현할 수 있습니다. 프로세서의 벤치마크를 추적하면 플래시 크기가 더 작은 더 저렴한 디바이스를 사용할 수 있습니다.
따라서 이 두 요소 모두가 비용 최적화에 기여합니다.

CoreMark는 속도 벤치마크이지만 광범위한 접근 방식 덕분에 크기에 대해서도 훌륭한 벤치마크입니다. 다양한 디바이스에서 벤치마크의 한 파일(coremark.c)을 살펴보면 그림 2에 표시된 바와 같이 사용된 디바이스에 따라 크기에 약간의 변동이 있음을 알 수 있습니다. 왼쪽에는 툴인 ARM RealView Compiler, GCC/GNU Tools ARM Embedded (LTO 포함 여부에 관계없음), ARM용 IAR ANSI C/C++ 컴파일러가 열거되어 있습니다. 막대에는 다양한 NXP 디바이스(Kinetis K70(Cortex-M4F), KL25Z(Cortex-M0+) 및 LPC11U24(Cortex-M0) 와 ST 디바이스 (STM32F207(Cortex-M3) 및 STM32F746(Cortex -M7))에서 코드 크기가 바이트 단위로 표시됩니다.

IAR Embedded Workbench(그림에서 ICARM V7.70.1)는 다른 툴보다 가변성 정도가 훨씬 작지만 유사한 비율의 크기 절감을 나타냅니다. 그림 3의 벤치마크에서 매트릭스 조작 코드를 보면 데이터가 유사합니다. 여기에서 IAR의 툴 세트로 컴파일된 코드가 다른 것보다 더 작다는 것을 다시 확인할 수 있습니다.

실제로 CoreMark의 34개 모듈 중 30개에서 ARM용 IAR Embedded Workbench가 더 작은 코드를 생성하며 전체적인 크기 차이는 약 20%입니다. 마찬가지로 실제 고객 애플리케이션을 사용하여 RX용 IAR Embedded Workbench와 RL78용 IAR Embedded Workbench에 대한 벤치마크를 조사해 보면 GCC 및 기타 툴보다 코드 크기가 27~28% 더 작습니다.

그림 2. 개발 툴 및 디바이스에 따른 크기(바이트) 가변성

(출처: IAR Systems)

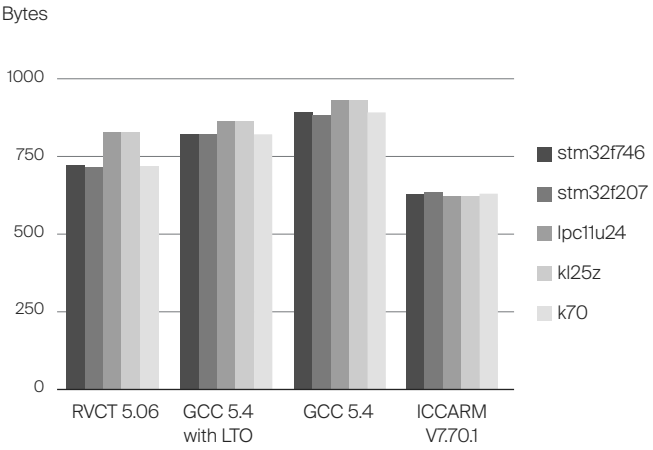
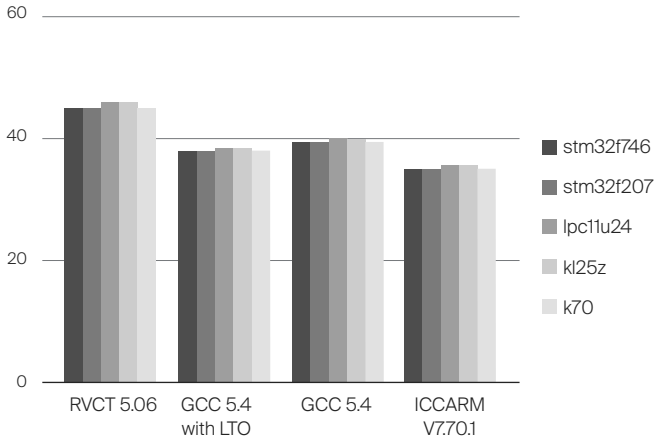


그림 3. 툴체인에 따른 크기 가변성

(출처: IAR Systems)



더 작은 디바이스 메모리 크기를 선택하면 얼마나 많은 비용을 절약할 수 있을까요? 분명히 이것은 기본 아키텍처, 주변 디바이스 세트와 패키지 유형이 유사한 더 큰 디바이스를 얻을 수 있는지 여부를 포함하여 많은 변수에 따라 달라집니다. 예를 들어, 동일한 제품군 및 실리콘 공급업체의 일부 인기 있는 Cortex-M4 디바이스를 살펴보겠습니다.

정확히 동일한 MCU와 주변 디바이스를 고려해 보면 주요 유통업체에서 단일 수량으로 $17.34(2022년 11월 기준)에 판매하는 512kB 플래시가 포함된 디바이스와 동일한 공급업체서 단일 수량으로 $21.47에 판매하는 1024kB 플래시가 포함된 유사한 디바이스의 경우 부품당 $4.13의 차이가 있습니다. 코드를 더 작은 디바이스에 맞출 수 없다면 그렇지 않은 경우보다 실리콘에 23.8% 더 많은 비용을 지출해야 합니다.

10,000개의 소량 생산에서도 추가 비용은 $41,000입니다. ST, NXP, Renesas, Microchip, TI와 같은 다양한 실리콘 공급업체의 다양한 벤치마크를 수행하면 ARM 코어 또는 독점 코어에서 256kB에서 512kB 또는 1025kB로 점프할 때 최소 $1.00의 가격 차이가 있습니다. 다시 말해 10,000개의 소량 생산에서 추가 비용은 최소 $10,000입니다. 가격 차이는 아키텍처와 실리콘 공급업체에 따라 약간 다를 수 있지만 총 비용 절감 규모는 상당할 수 있습니다.

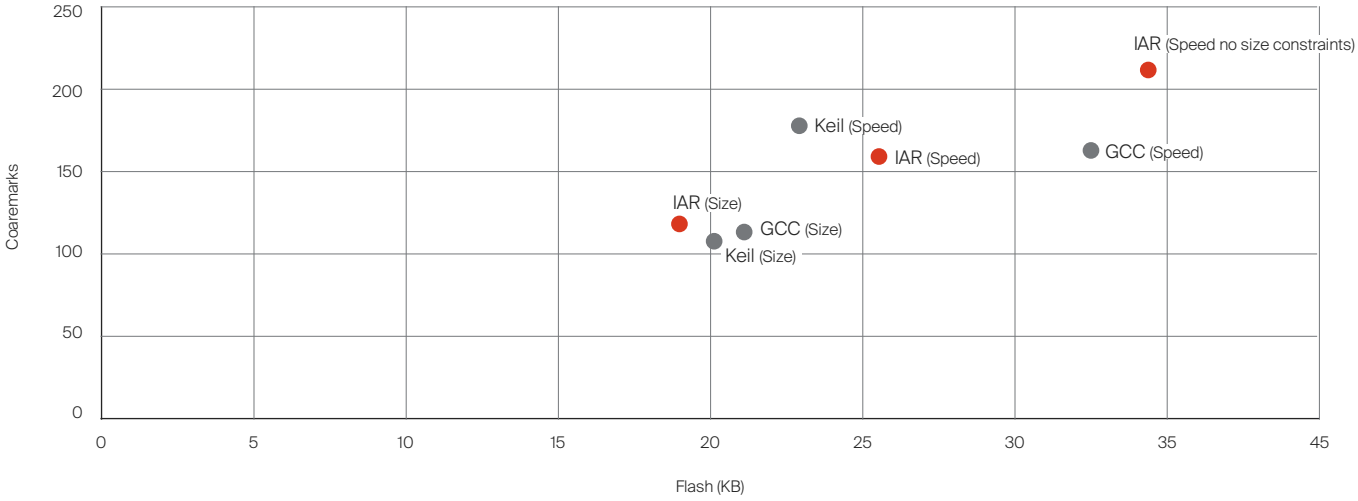
IAR Embedded Workbench를 다운로드하여
얼마나 코드 사이즈를 줄일 수 있는지 확인하세요.

다운로드 →

2. 코드 성능

애플리케이션 성능이 BOM(재료 명세서)에 어떤 영향을 미칠 수 있을까요? IAR Embedded Workbench를 사용하면 GCC 기반 툴 대비 얼마나 많은 성능 향상을 기대할 수 있을까요?

Performance (Coremarks) vs. Code size (Flash)



다시 말해, CoreMark 벤치마크는 매트릭스 조작, CRC 계산, 목록 처리(찾기 및 정렬 모두) 등 개발자가 수행하는 보다 일반적인 작업 중 일부를 통합하려고 하기 때문에 훌륭한 참조가 됩니다. 또한 컴파일러가 수행할 수 있는 작업에 대한 "실제" 비교를 제공하고 컴파일러 공급업체가 CoreMark 코드를 "수작업으로 최적화"하여 속이지 않도록 하는 변조 방지 메커니즘도 있습니다. 다양한 MCU 및 컴파일러 조합에 대한 CoreMark 벤치마크가 EEMBC 웹 사이트에서 제공되지만 Nordic Semiconductor에서 수행한 몇 가지 구체적인 벤치마크를 살펴보겠습니다.

성능만을 위해 컴파일될 때 IAR Embedded Workbench는 다른 툴체인들과는 많이 떨어져 있습니다. 4에서 볼 수 있듯이 특히 GCC와 실제로 큰 차이가 있음을 볼 수 있습니다.

이러한 벤치마크에서 IAR Embedded Workbench가 Keil 툴체인보다 19.1%, GCC 툴체인보다는 무려 29.8% 더 뛰어난 성능을 보인다는 것을 알 수 있습니다. CoreMark 웹 페이지에서 현재 및 최신 점수를 확인하는 것이 좋습니다. 벤치마크를 직접 실행하여 정확한 수치를 얻을 수도 있습니다.

그러나 디바이스 비용을 제외하고 이러한 유형의 최적화가 일반 개발자에게 정말 큰 의미가 있을까요? 이 문제에 유의해야 하는 이유를 이해하기 위해 크기 최적화를 조사했을 때와 유사한 분석을 수행해 보겠습니다.

그림 4. 성능 벤치마크

(출처: Nordic Semiconductor)

Complier	Coremarks	Coremarks/MHz	Code size (Flash, RAM) Bytes
IAR 7.30.4	**212.0**	**3.31**	**Flash: 35449, RAM: 2273**
ARMCC V5.17	**178.0**	**2.78**	**Flash: 23600, RAM: 1672**
GCC 5.2.1	**163.47**	**2.55**	**Flash: 33336, RAM: 2824**

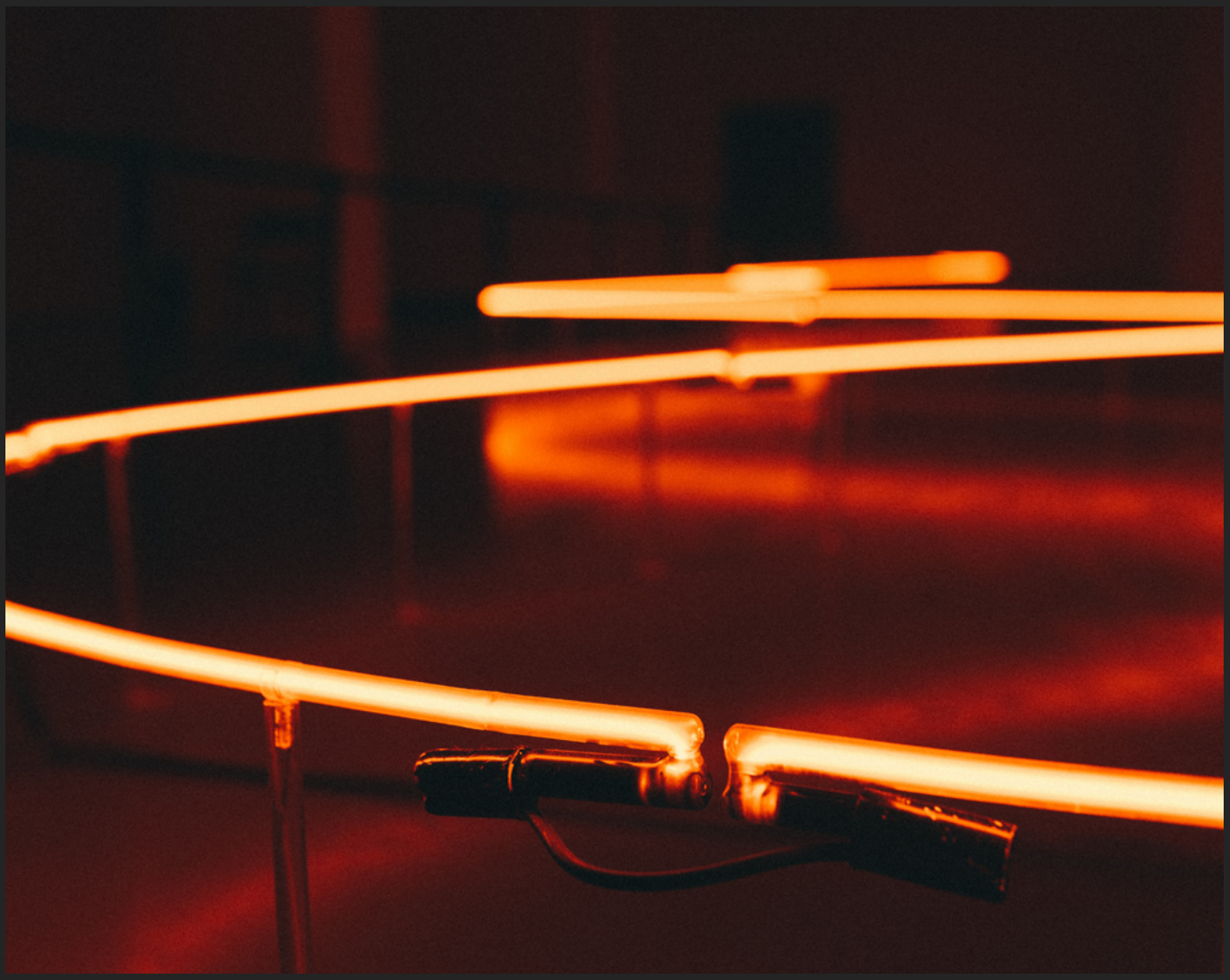
이전에 우리는 덜 효율적인 컴파일러를 사용하는 경우 더 많은 코드를 허용하기 위해 하나의 디바이스에 더 큰 플래시 공간이 있다는 점을 제외하고 정확히 동일한 두 디바이스를 살펴보았습니다. 대부분의 파라메트릭 검색에서는 최대 클럭 속도로 검색할 수 없기 때문에 디바이스의 최대 클럭 속도로 유사한 분석을 수행하는 것은 조금 까다롭습니다.

그러나 같은 실리콘 공급업체 제품군으로 패키징, 플래시 크기 및 RAM 크기, 32비트 타이머 수, D/A 변환기 수 등이 동일한 유사한 Cortex-M4 디바이스를 비교하면 통신 인터페이스(예: I2C 및 SPI의 수)가 약간 다를 수 있지만 주요 차이점은 최대 클록 속도(100MHz 대 180MHz)입니다.

그렇다면 180MHz는 100MHz보다 얼마나 더 비쌀까요? 주요 공급업체에 따르면 꽤 차이가 있습니다. 10,000개 수량 기준으로 180MHz 디바이스의 가격은 부품당 $3.78이며 100MHz 디바이스는 부품당 $2.89입니다 (2022년 11월 기준). 이는 애플리케이션에 필요한 성능을 얻기 위해 더 큰 클럭 속도까지 올려야 할 경우 30.8%의 차이를 의미합니다. 10,000개 생산 시 $9,000 이상의 차이가 납니다. 보다시피 속도 최적화는 BOM에 훨씬 더 큰 영향을 미칠 수 있으며, 특히 대량 생산의 경우에는 더욱 그렇습니다.

3. DevOps

생산성 향상을 위한 컴파일 시간 단축. 일반적으로 코드 라인을 추가하거나 소프트웨어를 수정하면 최악의 경우 최신 개발 워크플로우에서 소프트웨어 프로젝트 전체를 다시 빌드하게 됩니다. 이 경우 코드가 너무 많으면 빌드하는 데 시간이 오래 걸리고 결과적으로 이러한 대기 시간만큼 개발 기간이 늘어납니다. 따라서 개발과 관련된 빌드 시간을 줄이고 개발 팀의 생산성을 향상시키는 것이 특히 중요합니다. 여분의 시간은 더 많은 코드를 작성하거나 (코드 생성을 위해) 디자인 툴을 사용하거나 소스 코드의 품질에 집중하는데 사용하여 결과적으로 개발 조직의 효율성을 높일 수 있습니다.



이것이 귀사에 어떻게 적용될까요?

Steve McConnell의 저서 "Software Estimation: Demystifying the Black Art"에는 Cocomo II(Constructive Cost Model) 추정 모델에서 파생된 차트가 포함되어 있습니다. 이 차트에는 노력이 작업 시간(man-months) 대 SLOC(Line of Code)로 표현된 프로젝트 크기로 표시되어 있습니다. COCOMO II 노력 방정식을 조사하면 다음과 같습니다.

Effort = 2.94 * EAF * (KSLOC)E

여기서

EAF: 비용 요인에서 파생된 노력 조정 계수

E: 5개 규모의 요인에서 파생된 지수

KSLOC: 수천의 SLOC단위로 측정됨

노력 방정식의 EAF는 단순히 프로젝트의 각 비용 요인에 해당하는 노력 승수를 곱한 것입니다.

그림 5의 COCOMO II - Model Definition Manual에서 추출한 비용 요인을 살펴보면 상당한 비중을 차지합니다.

그림 5. LTEX(Language and Tool Experience) 및 소프트웨어 툴(TOOL) 사용 *(출처: Rose-Hulman Institute of Technology)*

LTEX 비용 요인						
LTEX 디스크립터:	≤ 2 개월	6 개월	1 년	3 년	6 년	-
등급 수준	매우 낮음	낮음	보통	높음	매우 높음	특히 높음
노력 승수	1.20	1.09	1.00	0.91	0.84	-

TOOL 비용 요인						
TOOL 디스크립터	편집, 코딩, 디버그	단순한 프론트엔드, 백엔드 CASE, 약간의 통합	적당히 통합된 기본 수명주기 툴	적당히 통합된 강력하고 성숙한 수명주기 툴	프로세스, 방법, 재사용과 잘 통합된 강력하고 성숙하며 능동적인 수명주기 툴	-
등급 수준	아주 낮음	낮음	공칭	높음	매우 높음	특히 높음
노력 승수	1.17	1.09	1.00	0.90	0.78	n/a

최악의 경우 매우 낮은 등급 수준에서 EAF(노력 조정 계수)= 1.40(1.20*1.17)이며, 최상의 경우 매우 높은 등급 수준에서 EAF = 0.66(0.84*0.78)입니다.
이는 전체 개발 팀의 생산성에 직접 영향을 미칩니다. 조직에 미치는 영향은 http://softwarecost.org/tools/COCOMO/ 에서 무료로 계산하고 조정할 수 있습니다. 이는 설계 및 코드 생성 툴에도 동일하게 적용됩니다. 자동 생성 코드의 빌드 시간이 길어지면 설계를 진행하기 전에 변경 사항이나 새로운 로직을 테스트하고 전체 시스템에 통합해야 하므로 설계 자체의 생산성에 영향을 미칩니다.

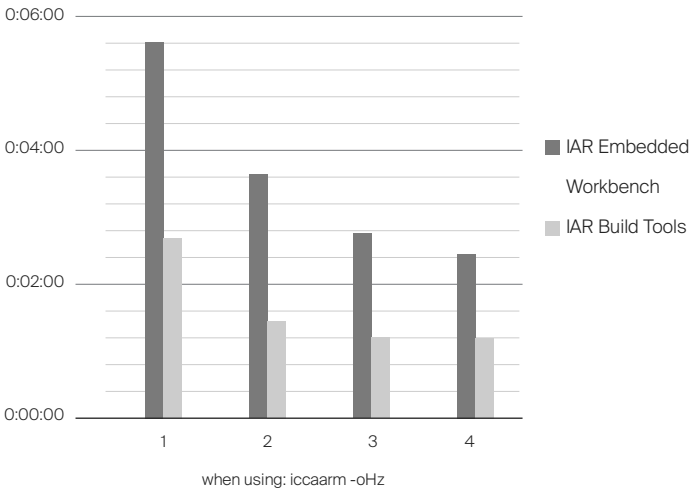
다양한 고객 피드백 및 고객 스토리에 따르면 IAR Embedded Workbench는 다른 상용 툴에 비해 빌드 속도가 최소 2배 이상 빠른 것으로 나타났습니다.
이는 IAR의 기능 안전 인증 툴도 마찬가지입니다. 크로스 플랫폼 지원을 위해 IAR Build Tools을 사용한 빌드 시간은 동일한 하드웨어 호스트를 사용하는 Ubuntu의 빌드 시간과 비교할 때 훨씬 더 나은 성능(4배 더 빠름)을 보여주었습니다. Ubuntu에서 표준 C-STAT 정적 분석 검사를 수행하는 데 걸리는 시간은 Windows에서 걸리는 시간의 25%만 소요되었습니다.

빌드 및 분석 결과가 더 빨리 전달된다는 것은 CD(Continuous Delivery, 지속적인 인도)의 단계로 더 빠르게 진행된다는 것을 의미합니다. 그림 6에 표시된 빌드 시간은 다음과 같습니다.

- 574개의 C/C++ 소스 파일
- 가장 높은 컴파일러 최적화 수준
- 프로젝트 구축 후 분석 수행
- 동일한 호스트 하드웨어(Intel i7-8700K, 24GB RAM)를 사용한 비교
- 1, 2, 4 및 8 CPU 코어 사용

그림 6. IAR Embedded Workbench 및 IAR Build Tools의 빌드 시간

(출처: IAR Systems)



다시 말해, 임베디드 소프트웨어 프로젝트의 빌드는 일반적으로 IAR Build Tools에서 수행하는 것이 더 빠르며 프로젝트를 빌드하는 데 걸리는 시간은 50% 미만입니다. 또한 최신 임베디드 개발 워크플로우에서 품질을 보장하고 지속적으로 빌드와 테스트를 실행하기 위해서는 자동화된 프로세스가 필수적으로 필요합니다. 임베디드 소프트웨어 R&D 팀은 크로스 플랫폼 프레임워크에서 기본 커맨드라인 툴의 동일한 기능으로 적절한 DevOps 방식을 구현할 때 새로운 기능의 출시 기간을 단축할 수 있습니다.

IAR 솔루션은 가상 머신, 컨테이너(Docker) 및 자체 호스팅 실행기를 포함한 CI/CD 파이프라인을 위해 Ubuntu, Red Hat 및 Windows에서 현대적이고 확장 가능한 빌드 서버 토폴로지를 지원합니다.

IAR Embedded Workbench를 다운로드하여 얼마나 생산성이 향상될 수 있는지 확인하세요.

다운로드 →

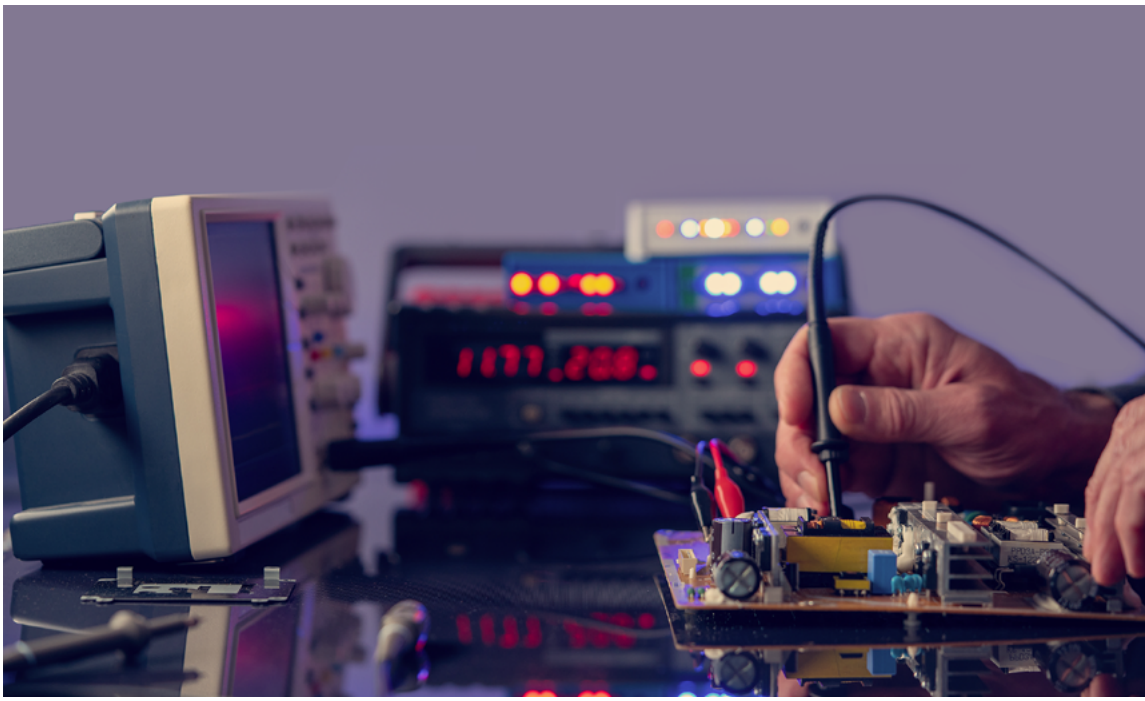
4. 디버깅

출시 시간을 줄이기 위한 디버깅 시간 단축.

그림 7. 스마트한 고급 디버깅 기능

(출처: IAR Systems)

Abdulaziz Alaboudi와 Thomas D. LaToza가 수행한 "디버깅 에피소드에 대한 탐색적 연구"에서는 개발자가 프로그래밍 시간의 약 절반을 디버깅에 사용하는 것으로 나타났습니다. StackOverflow와 Reddit의 토론에 따르면 개발자 시간의 최대 80% 또는 심지어 90%까지 디버깅에 소요되는 것으로 주장되고 있습니다. 이는 한 명의 개발자에 대해 연간 약 1,000~1,600시간에 해당합니다!
따라서 여러분의 개발자가 대단한 혁신에 시간을 소비한다고 생각할 수 있습니다. 하지만, 대부분의 예산이 디버깅에 사용되고 있다는 것을 다시 생각해보십시오.



디버깅이 너무 오래 걸리면 릴리스와 새 버전이 지연됩니다. R&D는 점점 더 복잡해지는 소프트웨어 시스템의 문제를 찾고 해결하는 데 점점 더 많은 시간을 소비하고 있습니다. 많은 프로그래머가 GDB와 같은 범용 디버거에 의지합니다. 이를 통해 프로그래머는 코드를 통해 조금씩 앞으로 나아가고 감시점을 설정할 수 있습니다. 이는 아마도 인류에게 가장 효율적인 디버그 방법일 것입니다. 불행하게도 임베디드 소프트웨어 개발자는 종종 툴 제한으로 인해 중단점 및 싱글 스테핑으로 디버깅하는 것이 기본입니다. 디버깅 시간을 줄이기 위해 개발자는 최신 마이크로 컨트롤러에서 사용할 수 있고 전문 개발 툴에서 지원하는 고급 디버깅 전략을 익혀야 합니다.

제품의 품질은 개발자가 사용할 수 있는 디버깅 기능에 따라 달라집니다. 여러분의 팀에서 특정 버그의 정확한 원인을 분석하고 추적할 수 있나요? 툴이 상세한 실시간 정보를 충분히 제공할 수 없어서 최선의 추측을 하여 문제를 수정하거나 주로 차선책을 적용하고 있나요?

IAR Embedded Workbench에 포함된 최첨단 C-SPY 디버거를 사용하면 애플리케이션을 실시간으로 완전히 제어할 수 있습니다.

또한 복잡한 중단점(코드 및 데이터 – 무제한), 감시점, 프로파일링, 코드 적용 범위, 인터럽트가 있는 타임라인, 전원 로깅, 추적과 같은 많은 스마트 기능도 제공됩니다(그림 7). 버그는 소스에서 쉽게 제거할 수 있어 디버깅에 소요되는 시간이 감소합니다.

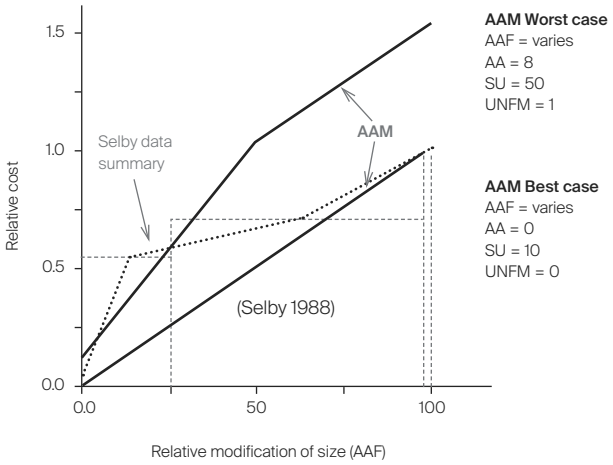
이러한 모든 기술을 익히고 사용 적절한 사용 시점을 숙지하면 시스템에 결함이 생겼을 때 디버깅에 소요되는 시간을 크게 줄일 수 있습니다. IAR Systems는 파트너사 개발자의 디버깅 시간이 80%에서 5% 미만으로 줄어든 사례에 대해 들었습니다. 보수적인 접근 방식을 취하여 디버깅 시간의 최소 2/3를 줄이는 것은 연간 최대 500시간을 의미하며, 재할당할 수 있는 많은 근무 시간(~1,000시간)을 확보하여 개발자의 개발 시간을 늘립니다.

5. 코드 품질

결함 비용. Steve McConnell의 저서 “Code Complete”에 따르면 평균적으로 개발자는 1,000줄의 코드당 70개의 버그를 생성합니다. 그 중 약 20%(코드 1,000줄당 15개의 버그)가 고객까지 전달됩니다. 그리고 씁쓸한 사실이지만 버그 한 개를 수정하는 것은 코드 한 줄을 작성하는 것보다 30배 더 오랜 시간이 걸립니다.

그림 8. Boehm의 COCOMO 비선형 재사용 효과 방법

(출처: Rose-Hulman Institute of Technology)



개발 주기 초기에 코드 품질 관리를 도입하면 오류의 영향과 그러한 오류를 제거하는 노력을 최소화할 수 있습니다. 코딩 표준이 잘 정의된 각 개발자의 컴퓨터에 바로 정적 분석을 제공하면 개발 중에 소스 코드에서 문제를 찾을 수 있어 오류 비용이 제품이 출시된 이 후 문제의 해결 비용보다 작습니다.

또한 많은 사람들이 재사용을 위한 코드 설계에 대해 이야기하지만 소프트웨어 추정 모델에서는 단순히 새 코드를 작성하는 노력의 50% 이상이 코드 재사용에 소요된다고 주장합니다.

그림 8에 나타난 Boehm의 COCOMO 방법은 점선에서 코드 작성의 상대적 비용이 재사용된 소프트웨어에 대한 수정에 따라 영향을 받는 방식을 추정합니다. x축은 재사용하려는 코드에 대한 수정 비율이고 y축은 새로운 코드를 작성했을 때의 비율을 나타냅니다. 코드의 3개 데이터 샘플 중 2개에 대해서는 재사용된 것으로 추정되는 코드의 많은 부분을 수정하여 코드를 처음부터 다시 작성하는 노력의 50%로 갑자기 점프할 필요가 없었습니다. AAM(Adaptation Adjustment Modifier, 적응 조정 수정자) 라인은 재사용된 제품의 작은 수정이 균형이 안 맞을 정도로 큰 비용을 발생시킬 수 있음을 보여줍니다.

그림 9. 총 비용 및 결함당 비용

(출처: Capers Jones: "Estimating Software Costs")

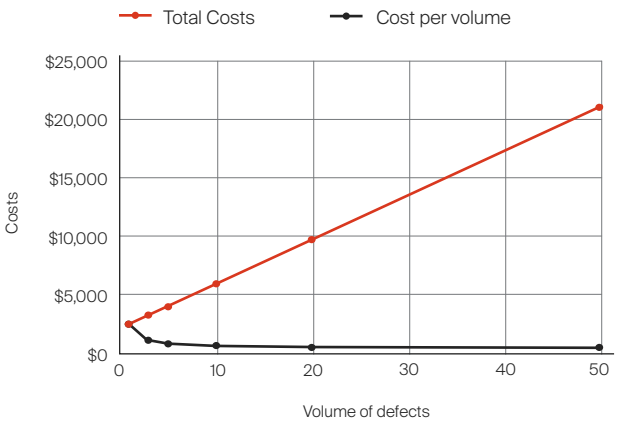
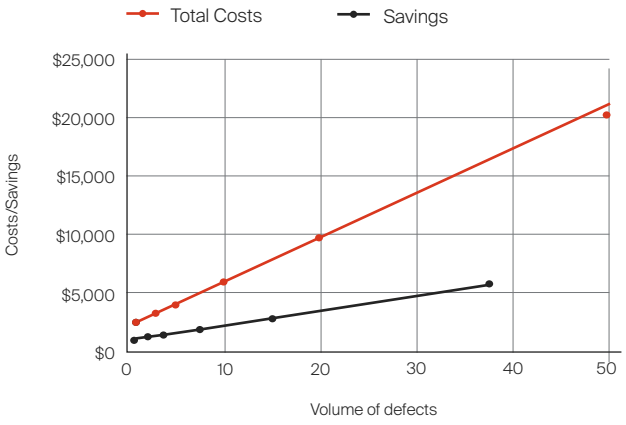


그림 10. 각 단계에서 테스트에 들어가는 결함의 수를 25% 줄이는 데 따른 총 비용 대비 절감액

(출처: Capers Jones: "Estimating Software Costs")



여기서 핵심 포인트는 코드를 정말로 재사용할 경우 비용 효율적이기 위해서는 코드의 품질이 매우 뛰어나고 순서대로 잘 설계되어야 한다는 것입니다.

코드 품질을 개선하는 가장 빠른 방법은 코드 분석 툴을 사용하는 것입니다. 실제로 기능적으로 안정된 인증 애플리케이션을 만들 경우에는 정적 분석을 사용해야 합니다. 이러한 유형의 툴은 코드에서 가장 일반적인 결함원을 찾는 데 도움이 되지만 개발자가 코드를 작성하려고 할 때, 특히 단지 무언가를 작동시키기 위해 테스트 코드를 올릴 때, 생각하거나 걱정하지 않는 경향이 있는 문제를 찾는 데도 도움이 됩니다. 이러한 유형의 툴은 코딩 표준을 적용하므로 더 나은 코드를 개발하는 데 실제로 도움이 됩니다.

정적 분석 솔루션의 품질에 따라 코드를 작성하는 동안 다른 많은 잠재적인 문제를 확인할 수 있습니다.

코드 품질이 큰 문제인 데에는 여러 이유가 있습니다. 첫째, 개발 조직의 성숙도에 따라 개발자는 시간의 최대 90%를 디버깅에 사용할 수 있습니다. 결함이 정식 빌드로 만들어지기 전에 신속하게 격리될 수 있다면 결함 주입률이 낮아져 조직의 품질 지표를 훨씬 더 빨리 충족할 수 있습니다. 둘째, 코드에 남아 있는 버그가 전반적으로 적다는 의미로서, 코드를 재사용하면 이전에 감지되지 않은 버그를 발견할 가능성이 낮아지기 때문에 코드가 재사용에 적합한 후보가 됩니다. 고품질 코드는 결함이 적기 때문에 유지 관리하기 더 쉽고 우수한 소프트웨어 엔지니어링 원칙을 따를 경우 확장이 더 쉬우므로 재사용하면 후속 프로젝트를 더 빠르게 진행할 수 있습니다.

품질이 중요한 이유

흥미로운 점은 각 단계에서 결함당 비용이 예상대로 증가하지만 결함의 양이 감소함에 따라 총 비용이 감소한다는 것입니다(Capers Jones의 저서 "Estimating Software Costs"의 그림 9 참조). 실제로 각 단계에서 버그를 찾아 수정하는 데 더 오랜 시간이 걸리지는 않지만 양이 줄었음에도 불구하고 비용은 여전히 존재합니다.

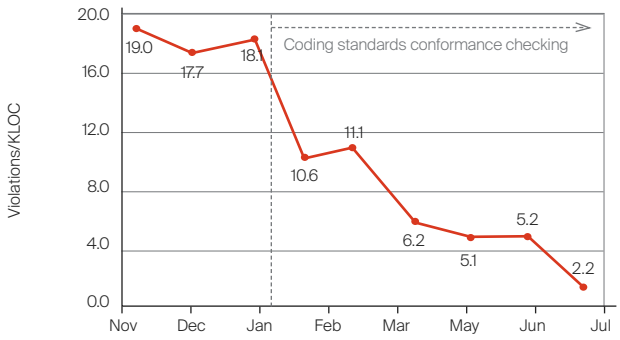
또한 제품이 운영 단계에 접어들면서 현장 제품 서비스의 영향으로 인해 결함당 유지보수 비용이 훨씬 더 높아진다는 점도 주목할 필요가 있습니다. 브랜드 손상, 미래 고객 및 소득 상실과 같은 다른 무형의 비용도 여전히 고려해야 할 요소입니다.

그렇다면 이러한 요소들이 고려된 ROI는 얼마일까요? 정적 분석을 사용하면 모든 개발 단계에서 소프트웨어 개발의 오류 수를 줄일 수 있습니다. 간단한 분석은 그림 10의 데이터를 사용하여 오류 수를 줄이는 것입니다. 개발 중에 도입된 이러한 오류 감소를 감안할 때 상당한 비용 절감을 볼 수 있습니다.

이러한 간단한 분석으로 버그당 약 126달러를 절약할 수 있으며, 개발 중에 코드 1,000줄당 평균 15개의 버그가 발생한다고 가정하면 코드 1,000줄당 1,900달러를 절약할 수 있습니다. 물론 인건비, 결함 감지 및 수리 시간, 결함 밀도와 같은 다른 요인에 따라 결과가 달라질 수 있습니다. 그러나 많은 시스템이 10~100KLOC 이상을 사용하기 때문에 정적 분석에 대한 비즈니스 사례는 명확합니다.

그림 11. 위반/KLOC

(출처: Dr. Dobbs)



코딩 기술 향상

게다가 Dr. Dobbs가 실시한 또 다른 연구(그림 11)에서는 결함 주입이 41%까지 낮아지는 것으로 나타났으며, 테스트 시간이 크게 절약되어 엔지니어링 시간이 절약될 뿐만 아니라 출시 시간이 단축되는 결과로 이어집니다.

이 연구에서 주입률은 조직에 코딩 표준이 도입될 때까지 매달 상당히 일정했고 이후 불량률이 크게 떨어졌습니다. 개발자가 표준에 익숙해지고 편차가 줄어들면서 불량률이 급감했습니다.

Google은 ACM 간행물에 코드 분석의 장점을 살펴보는 다음 글을 게시했습니다. 이 글은 C, C++ 및 Java를 포함한 전체 코드베이스에 대한 전체론적 관점을 취하지만 결과는 매우 명확합니다.

"컴파일러 오류는 개발 프로세스 초기에 표시되고 개발자 워크플로우에 통합됩니다. 우리는 컴파일러 검사 세트를 확장하는 것이 Google에서 코드 품질을 개선하는 데 효과적이라는 사실을 알게 되었습니다."

저자는 정적 분석 검사 내용을 컴파일러 워크플로우로 이동하고 오류로 표시되게 함으로써 툴의 결과에 대한 관심이 크게 향상되었으며 궁극적으로 코드의 품질이 훨씬 높아졌다고 말했습니다. 또한 최근에 컴파일 시간 오류가 발생한 개발자와 동일한 문제에 대한 수정 패치를 받은 개발자에게 보낸 설문 조사에 대해 논의합니다.

"컴파일 시간에 플래그가 지정된 문제(체크인 코드용 패치와 반대)를 인식하는 Google 개발자가 더 중요한 버그를 잡아냅니다. 예를 들어 설문 조사 참가자는 컴파일 시간에 플래그가 지정된 문제의 74%를 "실제 문제"로 간주한 반면 체크인 코드에서 발견된 문제에 대해서는 21%였습니다."

또한 이 기사는 정적 분석 툴을 통해 자동으로 커밋을 실행하고 엔지니어로 하여금 분석 대시보드를 보도록 했을 때 이를 따르는 엔지니어가 거의 없다고 말함으로써 워크플로우의 일부로 코드 분석을 수행하는 것 중요하다고 지적합니다. 컴파일 프로세스에서 즉각적인 피드백을 받으면 정적 분석을 더 쉽게 사용할 수 있고 무시하기가 더 어려워집니다. 따라서 Google은 모든 사람의 워크플로우에 정적 분석을 기본으로 통합하기로 결정했습니다.
그들은 코드 분석 툴이 성공하려면 개발자가 툴 사용에서 이점을 느끼고 즐겨야 한다고 믿습니다. 이 글의 요점은 코딩 표준이 실제로 개발 노력에 영향을 미친다는 것입니다.

6. 지원에 대한 액세스

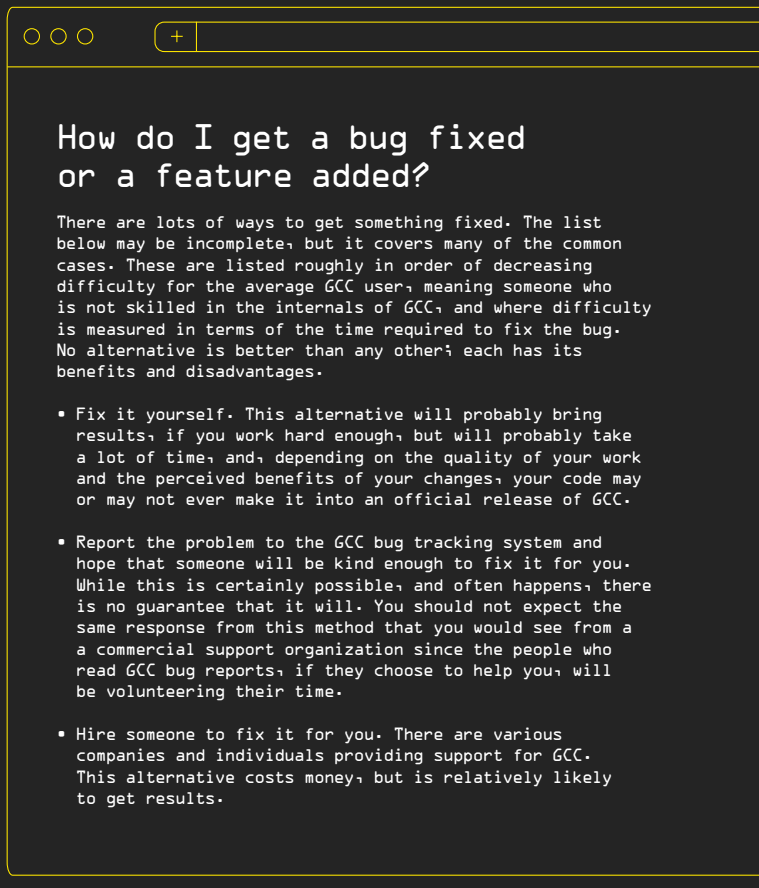
기술 지원에 대한 액세스로 개발 툴의 성과 향상.
전문 툴의 품질은 제공되는 기술 지원의 품질에 의해 사실상 차별화됩니다.

컴파일러나 라이브러리에 있는 버그와 같이 무료 툴에 문제가 있는 경우 사용자가 할 수 있는 유일한 일은 직접 수정하거나 해당 저장소에 문제를 게시하는 것입니다. 그들은 GNU 커뮤니티가 문제를 해결하거나 그림 12에 표시된 것과 같이 누군가 해결하거나 기능을 추가하도록 희망합니다. 이로 인해 사용자에게 실제로 어떤 비용(시간과 금전)이 들지는 예측할 수 없습니다.

개발 툴의 문제로 인해 전체 개발 팀의 운영을 중단할 필요가 없다는 것은 IAR Systems와 같은 공급업체의 전문 개발 툴을 사용하는 가장 큰 이점 중 하나입니다. IAR Systems는 영어, 스웨덴어, 독일어, 한국어, 일본어, 중국어, 아랍어와 같은 다양한 언어를 다루는 전 세계 현지 지원 팀과 함께 접근하기 쉬운 기술 지원을 제공합니다. 고객은 문제 해결 시간을 명확히 할 수 있고, 대체 가능한 해결책을 제시 받을 수 있어 개발에만 집중할 수 있습니다.

IAR Systems는 합리적인 노력으로 오류를 해결하거나 수리 시간에 따라 대체 가능한 해결책 제공을 통해 개발 중 문제의 심각도를 낮춥니다. IAR Systems는 중대하거나 심각한 것으로 정의된 오류가 라이선스 사용자에게 큰 불편을 초래할 수 있음을 인식하고 있으며, 여기에 정의된 수리 시간에 관계없이 가능한 한 빨리 오류를 수정하기 위해 합리적인 최선의 노력을 다할 것입니다.

그림 13은 IAR의 응답 시간이 1~2일 이내이고 중요한 문제에 대한 수리 시간이 근무일 기준 15일 이내인 일반 SUA(Support and Update Agreement, 지원 업데이트 계약) 고객의 수리 시간을 보여줍니다.



How do I get a bug fixed or a feature added?

There are lots of ways to get something fixed. The list below may be incomplete, but it covers many of the common cases. These are listed roughly in order of decreasing difficulty for the average GCC user, meaning someone who is not skilled in the internals of GCC, and where difficulty is measured in terms of the time required to fix the bug. No alternative is better than any other; each has its benefits and disadvantages.

- Fix it yourself. This alternative will probably bring results, if you work hard enough, but will probably take a lot of time, and, depending on the quality of your work and the perceived benefits of your changes, your code may or may not ever make it into an official release of GCC.
- Report the problem to the GCC bug tracking system and hope that someone will be kind enough to fix it for you. While this is certainly possible, and often happens, there is no guarantee that it will. You should not expect the same response from this method that you would see from a a commercial support organization since the people who read GCC bug reports, if they choose to help you, will be volunteering their time.
- Hire someone to fix it for you. There are various companies and individuals providing support for GCC. This alternative costs money, but is relatively likely to get results.

그림 12. https://gcc.gnu.org/faq.html#support의 스크린샷

그림 13. 일반 SUA 내 응답 및 수리 시간

(출처: IAR Systems)

난이도	응답 시간	문제 해결 시간
매우 어려운 문제	근무일 기준 1일	근무일 기준 최대 15일 이내
어려운 문제	근무일 기준 1일	근무일 기준 최대 30일 이내
일반적 문제	근무일 기준 2일	다음 업데이트 또는 제품의 업그레이드 버전 출시 시점에 적용 되며, 최대 1년 이내
쉬운 문제	근무일 기준 3일	IAR이 안내하는 해결 시간

예를 들어 GCC/LLVM에 기반한 EMBECOM의 사례 연구 "컴파일러 비용 계산"에 따르면 포괄적인 지원 및 서비스 계약으로 인한 비용 절감액을 쉽게 계산할 수 있습니다. 즉, 툴체인 유지 관리에는 매월 0.25개월의 엔지니어 노력이 필요합니다. 일반적으로 고용 비용은 복리후생, 급여세 및 기업 책임 보험에 따라 급여의 1.25~1.4배입니다. 미국 컴파일러 엔지니어의 연간 평균 급여 117,000달러에 1.4배 하면, 유지 관리 또는 심각한 버그를 수정하는 데 매월 3,400달러가 든다는 계산이 나옵니다..

IAR Embedded Workbench 라이선스는 12개월 동안의 SUA과 함께 제공됩니다. 결과적으로 고객은 SUA를 유지하는 데 매년 자신의 월별 유지 관리 비용의 50%만 지불하면 됩니다. IAR 팀이 근무일 기준 1~2일 내에 해결 방법을 제공할 수 있기 때문에 개발 팀이 항상 생산적인 상태를 유지한다는 사실을 고려하지 않습니다.

7. 개발 환경

하나의 IDE에서 가장 광범위한 디바이스 지원으로 개발자의 생산성 향상.

오늘날의 전자 기기는 종종 8, 16, 32 및 64비트 어플리케이션의 포트폴리오를 포함하는 점점 더 많은 수의 임베디드 시스템을 요구하고 있습니다. 동시에 임베디드 애플리케이션은 점점 더 복잡해지고 정교해지고 있습니다. 더욱 차별화된 기능을 갖춘 신제품에 대한 갈망이 커짐에 따라 단일 제품의 출시 시간이 회사 전체의 성공을 위한 결정적인 요인이 될 수 있습니다.

IAR Systems의 임베디드 개발 툴은 200개 이상의 반도체 파트너사의 8, 16, 32 및 64비트 MCU/MPU를 포괄하는 15,000개 이상의 디바이스를 지원하며 전 세계적으로 약 100,000명 이상의 개발자에게 서비스를 제공하고 있습니다. 이것은 가장 광범위한 디바이스 지원이며 가장 강력한 생태계입니다. IAR Embedded Workbench는 고객이 단일 IDE와 환경에서 모든 주요 공급업체의 프로세서 간에 자유롭게 이동할 수 있도록 합니다 (그림 14). 그러나 각 아키텍처에는 여전히 별도의 라이선스가 필요합니다.

다양한 대상에 대한 포괄적 플랫폼과 공통 구성 요소를 통해 시장에서 가장 광범위한 디바이스 지원이 가능합니다. 또한 IAR은 아키텍처별 프로세서별 적응 및 최적화를 통해 개발자가 효율적이고 안정적인 코드를 생성하는 동시에 개발 효율성을 향상시킬 수 있도록 합니다.

개발 시간을 단축할 수 있는 효율성을 제공하는 통합 개발 환경을 제공하는 컴파일러를 선택하는 것이 무엇보다 중요합니다. 그리고 이것은 일관된 애플리케이션 안정성을 보장하는 핵심 요소입니다. IAR Embedded Workbench 와 같은 임베디드 개발 툴로 표준화하면 개발 팀이 간소화된 개발 프로세스, 중단 없는 워크플로우, 모든 구성 요소가 원활하게 통합되는 단일 툴 상자를 사용할 수 있습니다.

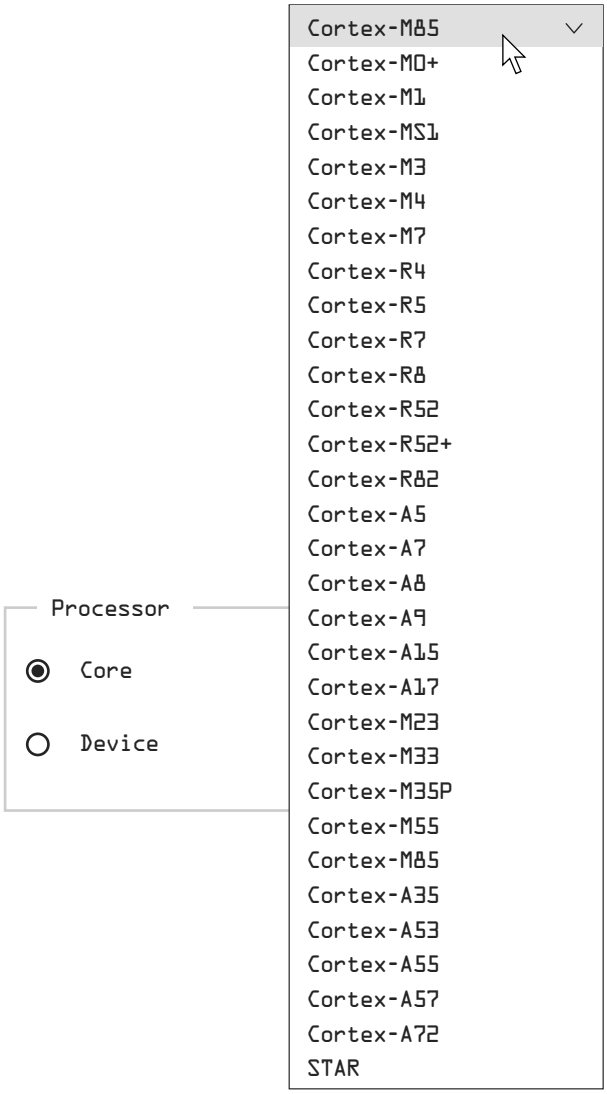


그림 14. 가장 광범위한 디바이스 지원을 특징으로 하는 단일 IDE 및 환경

(출처: IAR Systems)

또한 개발을 간소화하고 프로젝트와 프로세서 전반에서 코드를 재사용할 수 있어 생산 지연이 방지됩니다. 이것이 왜 중요할까요? 일반적으로 개발 팀은 여러 다른 프로세서와 동시에 작업합니다. 그들은 당면한 애플리케이션에 가장 적합한 프로세서를 선택할 수 있어야 합니다. 그리고 어떤 이유로 프로세서를 변경해야 하는 경우 처음부터 시작할 필요가 없고 툴도 변경하지 않아도 되므로 큰 이점이 있습니다.

장기적으로 성과를 거둘 수 있는 초기 투자가 있습니다. 시작을 위해 개발자가 새로운 IDE 및 툴체인을 배우는 데 최소 1주가 걸립니다.

그림 15. IAR 아카데미 포털 *(출처: IAR Systems)*

IAR Academy(그림 15)의 "IAR Embedded Workbench 시작하기 및 효율적인 프로그래밍과 디버깅"과 같은 대면 교육 과정은 개발자가 툴 모음의 기능을 최대한 활용하는 데 도움이 되며 근무일 기준으로 3일이 소요됩니다. 또한 개발자는 정보를 소화하고 스스로 사용 지침서를 시도하는 데 2일이 더 필요합니다. 미국 내 임베디드 개발자의 평균 급여 $104,000 x 1.4(복리후생, 급여세, 기업 책임 보험을 포함한 급여 비용)를 감안할 때 이는 개발자 한 명당 새로운 툴체인으로 속도를 높이기까지 거의 $2,800의 비용으로 해석됩니다.

이는 개발자가 새로운 툴체인에 대한 확신을 가지고 애플리케이션의 일관성과 안정성이 확보될 수 있기까지 시간이 더 오래 걸릴 수 있다는 점을 고려하지 않은 것입니다. 그러나 개발자 교육을 위한 이러한 일회성 비용은 가치 있는 투자로서 다른 프로세서를 사용한다고 해서 다시 지출할 필요가 없습니다. 특히 친숙한 툴체인을 사용하여 작업하는 개발자의 경험과 전문성이 늘어나면 시간과 비용을 더 많이 절약할 수 있습니다. 또한 최신 개발 워크플로우에는 유연성 및 생태계 파트너 솔루션과의 추가적인 통합이 요구됩니다. IAR Visual Studio Code extensions 기능(그림 16)은 모든 최신 버전의 IAR Embedded Workbench 및 IAR Build Tools와 호환되며 개발자가 VS Code에서 프로젝트를 빌드하고 개발할 수 있도록 Visual Studio Code Marketplace에서 다운로드 및 설치할 수 있습니다. 고급 기능과 함께 고품질 IAR 빌드 툴체인을 직접 활용하기 위해 IAR Eclipse 플러그인에도 동일하게 사용 가능합니다. 이 확장 기능은 CMake와 같은 다른 빌드 시스템, 소스 제어 및 GitHub와 같은 버전 관리 확장에 사용되어 개발 요구 사항을 충족할 수 있습니다.

IAR Embedded Workbench와 IAR Build Tools에는 처음 12개월 동안 전문 기술 지원에 대한 액세스 및 원활한 시작과 생산성 향상을 위한 IAR Academy on-demand 과정에 대한 액세스가 포함됩니다.

올인원 통합 IDE 및 확장 기능을 통해 프로그래머는 지속적으로 툴을 전환하지 않고도 한 곳에서 동일한 기능 세트를 사용할 수 있습니다. 개발 작업의 통합으로 개발자의 생산성과 효율성이 향상됩니다.

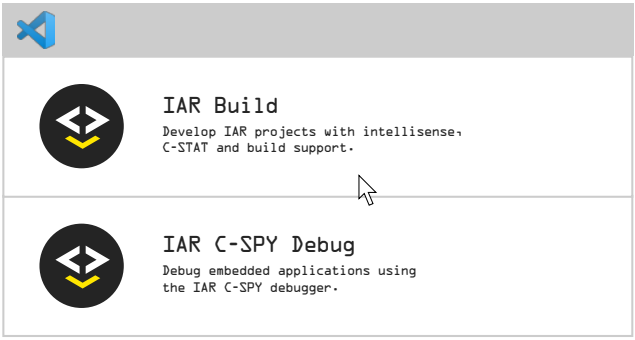


그림 16. VS Code용 IAR 빌드 및 IAR C-SPY 디버그 확장 *(출처: IAR Systems)*

8. 규정 준수 및 안전

인증된 툴을 사용하여 안전성 인증 과정 단축.
기능적 안전성은 어떠한 애플리케이션에서도 매우 바람직하지만
일부 애플리케이션에서는 절대적으로 필요합니다.

안전성 인증을 따를 경우 이점이 많습니다. 애플리케이션과 관련된 책임 위험을 줄이고 제품 리콜 가능성과 펌웨어 업데이트 횟수를 줄일 수 있습니다. 또한 회사의 명성과 기업 목표를 보호하는 것 외에도 국제적 표준과 요구 사항에 대한 준수를 보장합니다.

많은 표준과 안전성 인증이 시행되고 있습니다. 각각은 특정 산업 또는 제품 범주에 맞추어져 있습니다.
가장 광범위한 두 가지 인증 표준은 ISO 26262(도로 차량)과 IEC 61508(전자 안전 관련 시스템)이며, 이는 인증의 상위로 간주됩니다. 대부분의 기능적 안전성 개발 툴은 거의 모든 다른 인증 및 산업을 포괄하기 때문에 이 두 가지 표준에 따른 인증을 목표로 합니다.

특정 인증은 이 두 표준을 능가할 수 있지만 IEC 61508과 유사한 철도 시스템에 대한 EN 50128과 같이 다른 많은 표준의 기초로 참조됩니다. 일반적으로 모든 표준은 안전에 중요한 시스템의 위험을 평가하고 안전 목표를 할당하는 명확한 프로세스를 제공합니다. 또한 시스템 고장을 줄이기 위한 이상적 개발 프로세스 요건도 다룹니다. 마지막으로 제품 출시 후 기능적 안전성을 보장하기 위한 계속적인 절차도 있습니다. 요약하면, 표준은 위험을 식별하고 처리하는 방법을 설명하며 모든 표준에는 기능적 안전성에 대해 인증된 툴이 필요합니다.

넓은 범위의 안전 규적을 지원

Industrial
IEC 61508

Machinery control
ISO 13849 IEC 62061

Railway
EN 50128 EN 50657

Medical
IEC 62304

Agriculture & forestry
ISO 25119

Automotive
ISO 26262

Process industry
IEC 61511

Household appliances
IEC 620730

툴에 대한 기능적 안전성 인증은 코드를 컴파일할 때 개발 툴이 신뢰할 수 있고 반복 가능한 결과를 생성하도록 엄격한 검증 프로세스를 거쳤음을 의미합니다. 또한 다양한 기능적 안전성 표준에 명시된 구체적인 요건에 따라 툴이 작동하는 방식을 관리하기 위한 개발 프로세스가 마련되어 있으며 다양한 언어 표준을 준수하는지 확인하는 툴의 테스트 및 품질 조치가 있음을 의미합니다.

인증 절차는 다소 엄격합니다. IEC 61508 표준은 7.4.4 절에서 지원 툴의 검증 방법을 자세히 설명하지만 컴파일러의 자격에 대해서는 다소 모호합니다. 7.4.4.10절을 참조하면,

"선택한 프로그래밍 언어에는 해당하는 경우 국제 또는 국가 표준에 대한 평가를 포함하여 목적에 대한 적합성이 평가된 번역기가 있어야 합니다."

이런 저런 규정으로 인해 개발자 스스로 툴을 인증하기가 어려우며 적합성을 입증하기 위해 상당한 작업을 수행해야 하고 적합성이 입증되었다고 생각하는 이유를 문서화하기 위해 더 많은 작업을 수행해야 할 수 있습니다. 점점 더 높은 SIL(Safety Integrity Levels, 안전 무결성 수준)을 달성하려고 시도할수록 이는 더욱 어려워집니다.

프로세스의 일부는 일련의 검증 테스트 세트를 실행하는데, 여기서 수천 개의 테스트 프로그램이 컴파일되고 결과가 예상 결과와 비교됩니다. 또 다른 부분은 표준 준수 테스트입니다. 이러한 테스트 중 어느 것도 완전하지는 않지만 몇 가지 문제를 확인해야 합니다.

안전 검증의 요령은 모든 알려진 문제를 문서화하는 것입니다. 기능적 안정성이란 알려진 결함이 명확하게 문서화되고 결함을 확인하고 문서화하는 프로세스가 있음을 의미합니다. 전체 유효성 검사를 위해서는 예상되는 동작의 모든 편차를 수정하거나 문서화하고 정당화함으로써 합당하지 않은 알려진 편차가 없어야 합니다.

기능적 안전성를 위해 툴체인을 검증하는 것은 비용과 시간이 많이 듭니다. 툴 인증에는 최대 12개월이 소요될 수 있으며 명목상 2~4명의 직원이 필요합니다. 미국 내 임베디드 개발자의 급여를 고려할 때 예상 비용은 추가 테스트 요구 사항에 따라 최대 $145,000가 될 수 있습니다. 실제 수치는 필연적으로 프로젝트에 필요한 SIL에 따라 달라집니다.

결국 인증을 획득한 다른 프로젝트의 인증되지 않은 툴을 재사용하려는 경우에는 새 프로젝트가 이전 프로젝트와 충분히 유사하다는 것을 증명해야 합니다. 소스 코드 수준의 액세스 없이는 불가능한 이전 프로젝트와 동일한 툴체인 기능을 사용한다는 증거를 제공해야 합니다. 또한 안전성 인증을 받은 것과 동등한 방식으로 툴체인을 사용하고 있음을 증명해야 합니다. 일반적으로 툴을 재인증하기 위해서는 동일한 작업을 수행해야 할 수 있습니다.
기능적 안정성이 이미 인증된 개발 툴을 사용하면 더 이상 툴체인이 안전 표준을 준수하는지 증명할 필요가 없고 어플리케이션만 인증하면 됩니다. 실제로 기능적 안정성이 인증된 툴체인과 코딩 표준을 사용하면 애플리케이션 인증 프로세스가 쉽고 빨라지므로 많은 시간과 비용을 절약할 수 있습니다. 또한 이것은 SDLC(Software Development Lifecycle, 소프트웨어 개발 수명주기) 중 테스트-수정 단계에서 컴파일러 문제로 인해 문제가 발생하는지 궁금해하는 대신 소스 코드에서 버그를 찾는 데 집중할 수 있음을 의미합니다.

IAR Systems의 기능 안전 솔루션에는 10가지 안전 표준에 대해 TÜV SÜD에서 인증한 툴이 포함되며 특별한 기능 안전 계약을 통한 장기적 지원과 계약 기간 동안의 안전성 인증서 갱신이 함께 제공됩니다.
안전에 중요한 소프트웨어를 사용하는 고객을 위해 IAR은 기능적 안정성 SUA를 통해 우선 지원을 제공하며, 하루 이내에 응답하고 영업일 기준 10일 이내에 중요 문제를 해결합니다(그림 17).

그림 17. 기능적 안전성 SUA 내 응답 및 수리 시간

(출처: IAR Systems)

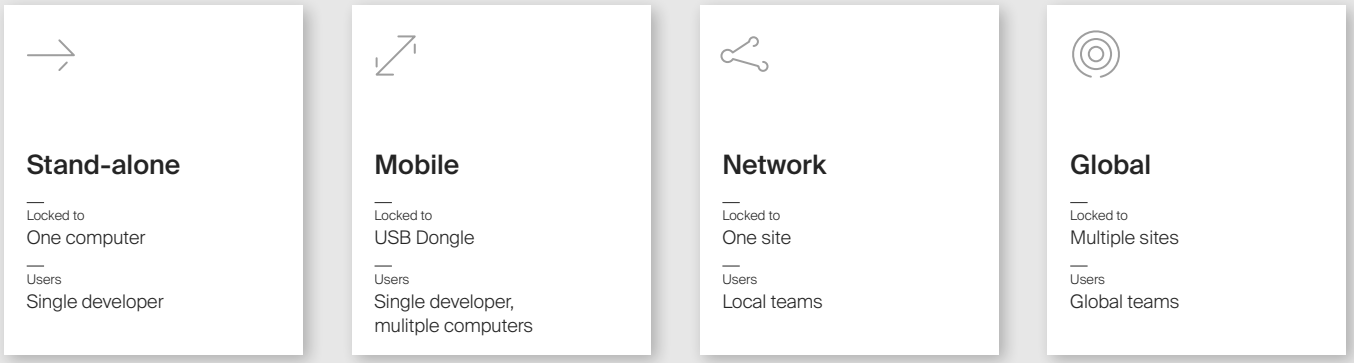
난이도	응답 시간	문제 해결 시간
매우 어려운 문제	근무일 기준 1일 이내	근무일 기준 최대 10일 이내
어려운 문제	근무일 기준 1일 이내	근무일 기준 최대 20일 이내
일반적 문제	근무일 기준 1일 이내	다음 업데이트 또는 제품의 업그레이드 버전 출시 시점에 적용되며, 최대 1년 이내
쉬운 문제	근무일 기준 1일 이내	IAR이 안내하는 해결 시간

9. 라이선싱

필요에 맞는 라이선스 모델 찾기. 적합한 라이선스 유형을 찾으면 ROI를 극대화할 수 있습니다. 많은 라이선스 사용 사례가 있으며 라이선스를 올바르게 조합하고 관리하면 기본적으로 툴 지출을 최적화하는 데 도움이 됩니다. 모든 것은 조직이나 개발팀이 필요로 하는 것이 무엇인지에 대한 질문으로 돌아옵니다.

그림 18. 유연한 라이선스 유형 *(출처: IAR Systems)*

Learn more →



IAR Systems는 기업의 ROI를 극대화하기 위해 유연한 라이선스 제공 및 가격 옵션을 제공합니다. 독립형 라이선스, 모바일 라이선스, 네트워크 및 글로벌 라이센스에 이르기까지 다양한 라이센스 유형을 제공하여 라이센스 관리가 용이합니다. 라이센스 풀은 그림 18에 표시되어 있습니다.

네트워크 라이센스는 개발자 팀에게 있어 편리하고 비용 효율적입니다. 로컬 네트워크를 통해 사용자 그룹 간에 라이선스 풀을 공유할 수 있습니다. 동시 사용자 수에는 제한이 있지만 라이센스를 점유할 수 있는 설치된 라이선스 수에는 제한이 없습니다. 네트워크 라이센스는 라이센스 서버 관리 도구에서 관리합니다. 새로운 추가 사용자 수를 기존 네트워크 라이선스에 추가할 수 있습니다.

여러 현장과 다양한 국가에서 운영 및 개발 프로젝트를 수행하는 고객을 위해 IAR은 글로벌 네트워크 라이선스를 통해 지리적 유연성을 제공합니다. 일반적인 네트워크 라이선스는 단일 지리적 현장으로 제한되지만 글로벌 네트워크 라이선스의 경우 사용자는 전 세계 여러 현장에서 동일한 네트워크 라이선스에 액세스할 수 있습니다.

모바일 라이선스는 단일 사용자 라이선스로서 사용자가 작업 위치에 따라 유연하게 사용할 수 있습니다. USB 동글에 잠겨 있어 휴대가 가능하고 어떠한 PC에서도 사용할 수 있습니다. PC가 네트워크에 연결되어 있지 않아도 됩니다. 라이센스를 동글에 보관하면 개발 PC의 오류로부터 라이센스가 보호됩니다.

PC 잠금 라이선스(독립형)는 특정 PC에 잠겨 있습니다. 이것은 개인용 단일 사용자 라이센스로서 PC에 물리적으로 액세스할 수 있는 경우에만 사용 가능합니다. PC가 네트워크에 연결되어 있지 않아도 작동합니다. 가상의 회사에서 두 사이트에서 개발 툴에 액세스해야 하는 개발자가 30명(각 사이트당 15명)이며 프로젝트 기간이 2년(SUA 갱신의 경우 연간 20%)인 경우를 살펴봅시다.

가격 기준($)인 독립형 라이선스를 고려하면 이는 모든 현지 통화가 될 수 있으며 아키텍처(8, 16, 32 및 64비트)에 따라 다릅니다. 다른 라이센스 유형에는 유연성에 따라 프리미엄 비용이 있습니다. 동글 라이센스는 16%(1.16$), 네트워크 라이센스는 33%(1.33$), 글로벌 라이센스는 100%(2$) 추가 비용이 듭니다.

모든 개발자에게 독립형 라이선스를 제공하는 비용은 30 x $ = 30$입니다 (동글 라이선스의 경우 34.8$). 회사가 네트워크 라이센스로 이전할 경우에는 각 사이트에 있는 15명의 개발자를 포괄하는 10개의 네트워크 라이센스 (권장) 비용이 듭니다. 네트워크 라이센스는 로컬 사이트에만 허용되므로 사이트당 비용은 10 x (1.33$) = 13.3$이며 총 26.6$이며 독립형 모델에 비해 비용이 ~13% 감소합니다. 다음 단계는 글로벌 라이센스로 이전하는 것으로 두 사이트에 대해 총 12개의 라이센스(권장 사항이지만 사이트의 작업 시간이 겹치지 않는 경우 더 적을 수 있음)가 필요하여 12 x (2$) = 24$의 비용이 듭니다. 이것은 네트워크 라이센스에 비해 ~11%, 독립형 라이센스에 비해 ~25%의 비용 절감을 의미합니다. 이 예에서는 교체가 필요한 손상된 워크스테이션 또는 분실된 동글에 대한 관리와 문제가 고려되지 않습니다. 연간 SUA에 대한 20%도 비용 절감 비율을 따릅니다.

조직에 적합한 라이선스 유형을 찾으면 총 라이선스 비용의 최대 25%를 절감하고 라이선스 관리를 용이하게 하는 등 상당한 영향을 미칠 수 있습니다.

10. 결론

제품 개발 시간에 대한 지속적인 관리는 비용을 관리하고 납품 목표를 달성하는 데 매우 중요합니다. 이러한 목표는 엔지니어가 제품을 신속하고 효율적으로 만들 수 있도록 설계된 툴과 서비스를 사용하여 달성할 수 있습니다.

제품 설계에서 특정 디바이스를 사용할 목적으로 진입 장벽을 낮추기 위해 제공되는 “무료” 툴에 비해 초기 비용이 드는 상용 툴을 사용하는 것은 개발 일정을 유지하고 전체적인 제품 개발 비용을 줄이는 효과적인 방법이 될 수 있습니다.
상용 툴 사용으로 많은 기업과 개발팀이 더 빠른 출시 시간을 “확보”하고 우수한 품질의 제품을 제공하고 있습니다.
끝으로, IAR Systems에서 제공하는 것과 같은 전문 솔루션으로 작업하는 개발자를 위해 ROI 및 TCO에 대한 명확한 이해를 위하여 아래와 같이 사례를 정리하여 드립니다.

표 1. ROI 및 TCO 요약

(출처: IAR Systems)

사용 사례	절감 및 개선 사항	절감액
코드 크기와 벤치마크에 관심을 가져야 하는 이유는 무엇입니까?	디바이스당 $1 ~ $4	제품 시리즈당 $10,000 ~ $40,000
애플리케이션 성능이 BOM(재료 명세서)에 어떤 영향을 미칠 수 있습니까?	디바이스당 $0.50 ~ $0.90	제품 시리즈당 $5,000~$9,000
컴파일 시간 단축으로 생산성 향상	빌드 시간을 최대 50% 단축하여 생산성 향상(Linux)	클라우드 서비스에서 최대 50%의 인스턴스/시간 절약
디버깅 시간 단축으로 출시 시간 단축	작업자 시간 ~1,000시간 확보	연간 $24,000
결함 비용 및 코드 품질이 중요한 이유	1,000 LOC당 $1900	1,000~5,000KLOC당 $19,000~$95,000
기술 지원에 대한 엑세스로 개발 툴의 성과 향상	매월 $2,400	프로젝트당 $43,000
하나의 IDE에서 가장 광범위한 디바이스 지원으로 개발자의 생산성 향상	개발자당 $2,000	프로젝트당 $10,000
인증된 툴을 사용하여 안전성 인증 경로 단축	프로젝트당 최대 $145,000	프로젝트당 $145,000
필요에 맞는 라이선스 모델 찾기	사용 극대화 및 안정적인 투자 (개발자 수에 따라 다름)	총 라이센스 비용의 최대 25% 절감

저자

저자

Rafael Taubinger
Sr. Product Marketing Manager
(IAR Systems 소속)

기여자

Anders Holmberg
Chief Technology Officer
(IAR Systems 소속)

David Källberg
FAE Manager EMEA
(IAR Systems 소속)

Shawn Prestridge
FAE Manager US
(IAR Systems 소속)

Hyun-Do Lee
Sales Manager
(IAR Systems 소속)

IAR SYSTEMS

iar.com

Tomorrow’s intelligence,
delivered today